Quaderni di Statistica
Vol. 7, 2005

A fast algorithm for computing the sampling distribution of a statistic from discrete populations

Domenico Piccolo
Dipartimento di Scienze Statistiche, Università di Napoli Federico II
E-mail: domenico.piccolo@unina.it

Summary: In this work we propose a fast algorithm for computing the exact small sampling distribution of a given statistic, when the population random variable is discrete. The algorithm relies on a recursion on block matrices that describes all possible random samples that can be generated. In this way, the power of modern programming which defines objects in term of matrices is fully exploited for effective computations. Finally, some instances of applications are shown.

Keywords: Exact sampling distribution, Matrix languages, Numerical computations.

1. Introduction

The evaluation of the exact distribution of a statistic is a relevant target in statistical inference, with reference to estimators, test statistics and confidence intervals. As a matter of fact, the exact distribution of a statistic allows:

i) to evaluate the percentiles and confidence intervals;

ii) to assess the performance of many asymptotic results.

In fact, these problems are relevant when the size of the sample is small since, for moderate and large sample sizes, asymptotic results are

generally applicable. Thus, we limit our numerical developments only to sample of small size.

In this regard, we note that the sample space of all possible samples of size n increases exponentially with n. Thus, the programming of all its elements is a demanding task.

In the following, we refer to random samples that are generated by discrete random variables, and we investigate an algorithm that exploits the power of matrix languages in order to save time in the sequential looping. This capability may be easily applied to modern matrix languages (as GAUSS, R, MATLAB, for instance). Then, we implement the proposal in the first two environments and check the proposed procedure for some statistics. Some final comments conclude the paper.

2. Notation and formalization

Let $X \sim p(x;\theta)$ be a discrete random variable defined over a finite support which, in first instance, we suppose to be $\mathbb{S}_{\mathbb{X}} = \{x : x = 1, 2, \ldots, m\}$. In the final section, we will generalize it to $\mathbb{S}_{\mathbb{X}}{}' = \{x : x = x_1, x_2, \ldots, x_m\}$.

Then, $\underline{X}$ =$(X_1, X_2, \ldots, X_n)$ is a random sample of size n, so that $X_i \sim p_X(x;\theta)$, $i = 1, 2, \ldots, n$ is a collection of n identically and independently distributed random variables.

We let $\mathcal{C}_n(X)$ be the sample space, that is the collection of all the random samples $\underline{X}$ of size n generated by Bernoulli drawing from X.

For any well defined real function $T(.)$, we are interested in deriving the exact probability distribution of $T_n = T(\underline{X})$. Since $\mathbb{S}_{\mathbb{X}}$ is a discrete support, the mapping $\underline{X} \rightarrow T_n$ will define a discrete support $\mathbb{S}_{\mathbb{T}} = \{t : t = t_1, t_2, \ldots, t_{n(m)}\}$.

Thus, the statistic T_n will assume real values with some probability distribution $p_T(t;\theta)$, $\forall t \in \mathbb{S}_{\mathbb{T}}$. Indeed, we are looking for the exact derivation of $p_T(.;\theta)$.

For any observed sample $\underline{x} = (x_1, x_2, \ldots, x_n)$, we compute the

probability:

$$P_r\left(X_1 = x_1,\ X_2 = x_2,\ \ldots,\ X_n = x_n\right)$$
$$= P_r\left(T_n = T(x_1, x_2, \ldots, x_n)\ \right) = \prod_{i=1}^{n} p_X(x_i;\theta).$$

To obtain the distribution of T_n, we generate all the possible random samples of $\mathcal{C}_n(X)$ and compute for each of them both the value $t = T(x_1, x_2, \ldots, x_n)$ and the probability $P_r\left(T_n = t\right)$. In this way, $\forall t \in \mathbb{S}_{\mathbb{T}}$,

$$P_r\left(T_n = t\right) = \sum_{T(x_1,x_2,\ldots,x_n)\ =t} P_r\left(X_1 = x_1,\ X_2 = x_2,\ \ldots,\ X_n = x_n\right).$$

From an operational point of view, we need to generate all the samples in $\mathcal{C}_n(X)$, compute for each of them the joint probabilities and the values that the statistic $\mathrm{T}(x_1, x_2, \ldots, x_n)$ will assume; then, by summing the corresponding probabilities over the same values of $t \in \mathbb{S}_{\mathbb{T}}$, we get the exact probability distribution of the statistics T_n. It is immediate to verify that the resulting distribution is a well defined probability distribution[1].

3. An example

Suppose we draw a sample of size $n = 3$ from a discrete random variable X defined over a support $\mathbb{S}_{\mathbb{X}}$ with $m = 5$. Then, for generating all the samples in $\mathcal{C}_n(X)$, we need an algorithm that produces the $5^3 = 125$ samples obtained by changing the values that the 3 random variables in $\underline{X} = (X_1, X_2, X_3)$ assumes in the support of X, that is the number between 1 and 5.

If, for instance, we are interested in deriving the probability distribution of the *sample range* statistic $T_n = max(\underline{X}\ ) - min(\underline{X}\ )$, we have to define the related support $\mathbb{S}_{\mathbb{T}} = \{t :\ t = 0, 1, 2, 3, 4\}$ and compute the probabilities for each $t \in \mathbb{S}_{\mathbb{T}}$. Indeed, we need to implement an algorithm

[1] Of course, the approach can be easily generalized to sampling without replacement or sampling with unequal probabilities.

that will generate *all* the possible samples in $\mathcal{C}_n(X)$, in order to compute the related probabilities by exploiting the same loop.

This result could be simply obtained as follows:

```
for i (1,5,1);
   for j (1,5,1);
        for k (1,5,1);
        t_n(i,j,k) = max(i,j,k) − min(i,j,k);
        Pr(X_1 = i, X_2 = j, X_3 = k) =
        = Pr(X = i) Pr(X = j) Pr(X = k);
        endfor;
   endfor;
endfor;
```

Then, by selecting the distinct values of t_n and by summing the corresponding probabilities, it is immediate to derive the sampling distribution of T_n, for any well defined discrete random variable.

However, the time required for this loop is of the order m^n, that is unfeasible for any real sample sizes, although small.

To simplify the following discussion, we prefer to discuss in some detail what happens in the previous example, when $(m = 5; n = 3)$.

Then, we list all the $5^3 = 125$ elements of $\mathcal{C}_(X)$ in a matrix $\mathsf{A}(\mathcal{C}_n)$ where each row represents the observed sample. In each row, we list also the corresponding observed value of the statistic T_n.

Looking at the next matrix $\mathsf{A}(\mathcal{C}_n)$, it is immediate to realize that the third column vector is formed by expanding each member of the sequence $(1, 2, 3, 4, 5)$ for $5^0 = 1$ time and then replicating the resulting vector

of length 5 for $5^2 = 25$ times.

$$\mathsf{A}(\mathcal{C}_n) \;=\; \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 2 \\ 1 & 1 & 3 \\ 1 & 1 & 4 \\ 1 & 1 & 5 \\ 1 & 2 & 1 \\ 1 & 2 & 2 \\ 1 & 2 & 3 \\ 1 & 2 & 4 \\ 1 & 2 & 5 \\ 1 & 3 & 1 \\ 1 & 3 & 2 \\ 1 & 3 & 3 \\ \dots & \dots & \dots \\ \dots & \dots & \dots \\ 5 & 3 & 4 \\ 5 & 3 & 5 \\ 5 & 4 & 1 \\ 5 & 4 & 2 \\ 5 & 4 & 3 \\ 5 & 4 & 4 \\ 5 & 4 & 5 \\ 5 & 5 & 1 \\ 5 & 5 & 2 \\ 5 & 5 & 3 \\ 5 & 5 & 4 \\ 5 & 5 & 5 \end{pmatrix} \quad \Longrightarrow \quad \begin{pmatrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 1 \\ 1 \\ 2 \\ 3 \\ 4 \\ 2 \\ 2 \\ 2 \\ \dots \\ \dots \\ 2 \\ 2 \\ 4 \\ 3 \\ 2 \\ 1 \\ 1 \\ 4 \\ 3 \\ 2 \\ 1 \\ 0 \end{pmatrix}$$

Then, the second column vector is obtained by replicating each member of the sequences $(1, 2, 3, 4, 5)$ for $5^1 = 5$ times and then replicating the resulting vector of length $5^2 = 25$ for $5^1 = 5$ times.

Finally, the first column vector is obtained by expanding each member of the sequence $(1, 2, 3, 4, 5)$ for $5^2 = 25$ times and then replicating the resulting vector of length $5^3 = 125$ for $5^0 = 1$ time.

It is worth to note that each vector is formed by $5^3 = 125$ elements.

This example shows that all the possible samples in $\mathcal{C}_n(X)$ can be produced by implementing an algorithm able to provide a nested sequence of vectors of increasing size based on the original sequence in $\mathbb{S}_{\mathbb{X}} = \{x : x = 1, 2, \ldots, m\}$ and made up by repeating the single elements.

In order to reach this objective in a general form, in the next section, we will introduce some vector operators.

4. The algorithm in a matrix language

For any column vectors

$$\boldsymbol{v} = (v_1, v_2, \ldots, v_p)', \ \boldsymbol{a} = (a_1, a_2, \ldots, a_p)', \ \boldsymbol{b} = (b_1, b_2, \ldots, b_p)'$$

and any positive integers r and s, we define:

- **`Expand`** operator:

$$\mathcal{E}[\boldsymbol{v}]^r = \left(\underbrace{v_1, v_1, \ldots, v_1}_{r \text{ times}}, \underbrace{v_2, v_2, \ldots, v_2}_{r \text{ times}}, \ \ldots, \ \underbrace{v_p, v_p, \ldots, v_p}_{r \text{ times}} \right)' .$$

- **`Replicate`** operator:

$$\mathcal{R}[\boldsymbol{v}]^s = \left(\underbrace{v_1, v_2, \ldots, v_p}_{1}, \underbrace{v_1, v_2, \ldots, v_p}_{2}, \ \ldots, \ \underbrace{v_1, v_2, \ldots, v_p}_{s} \right)' .$$

- **`Joining`** operator:

$$\boldsymbol{a} \, \mathcal{J} \, \boldsymbol{b} = \begin{pmatrix} a_1 & b_1 \\ a_2 & b_2 \\ \ldots & \ldots \\ a_p & b_p \end{pmatrix} .$$

With an obvious notation, the last operator may be generalized:

$$\underset{i=1}{\overset{n}{\mathcal{J}}} \boldsymbol{a}_i = \boldsymbol{a}_1 \mathcal{J} \boldsymbol{a}_2 \mathcal{J} \ \ldots \mathcal{J} \boldsymbol{a}_n = \begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \ldots & \ldots & \ldots & \ldots \\ a_{p1} & a_{p2} & \ldots & a_{pn} \end{pmatrix}.$$

Of course, if $rows(\boldsymbol{v}) = p$, then

$$rows(\mathcal{E}[\boldsymbol{v}]^r) = r\,p; \qquad rows(\mathcal{R}[\boldsymbol{v}]^s) = s\,p.$$

Now, let us define by $\boldsymbol{u} = (1, 2, \ldots, m)'$ the column vector consisting of the first m positive integers. Then, the matrix $\mathsf{A}(\mathcal{C}_n)$ containing all the possible random sample of size n may be written as[2]:

$$\mathsf{A}(\mathcal{C}_n) = \left(\mathcal{R}[\mathcal{E}[\boldsymbol{u}]^{m^{n-1}}]^{m^0}\right) \mathcal{J} \ldots \mathcal{J} \left(\mathcal{R}[\mathcal{E}[\boldsymbol{u}]^{m^1}]^{m^{n-2}}\right)$$
$$\mathcal{J} \left(\mathcal{R}[\mathcal{E}[\boldsymbol{u}]^{m^0}]^{m^{n-1}}\right) = \underset{i=1}{\overset{n}{\mathcal{J}}} \left(\mathcal{R}\left[\mathcal{E}[\boldsymbol{u}]^{m^{n-i}}\right]^{m^{i-1}}\right).$$

This is the main result of our paper. Then, to check in a real situation the correctness of the previous expression, we could let $m = 5$; $n = 3$ and derive the results of the worked example that we have fully discussed in section 3.

It is worth to note that the required loop is of order n, lowering the time required for generating the elements of $\mathcal{C}_n(X)$. Now, if a matrix language is really effective for manipulating vectors as with the previous operators by means of intrinsic commands, we can deduce that passing from a loop of m^n iterations to a loop of n iterations one achieves a substantial reduction in computing time.

---

[2] We are writing m^0, $m^1, \ldots$ (instead of 1, $m, \ldots$) in order to stress the sequential pattern of the orders of the implied operators.

5. The implementation of the algorithm

We implement the proposed procedure in two of the most widespread languages in the statistical community[3]. Their ability derives from the possibility to manipulate matrices in an effective way.

First of all, we refer to the $GAUSS$ language, where we have the following correspondences[4]:

$$\begin{aligned}
\mathcal{E}[\boldsymbol{v}]^r &\Longrightarrow \texttt{reshape(vec(}\boldsymbol{v}'.*ones(r, rows(\boldsymbol{v}))), rows(\boldsymbol{v})*r, 1);\\
\mathcal{R}[\boldsymbol{v}]^s &\Longrightarrow \texttt{reshape(}\boldsymbol{v}, s*rows(\boldsymbol{v}), 1);\\
\boldsymbol{a}\,\mathcal{J}\,\boldsymbol{b} &\Longrightarrow \boldsymbol{a}\sim\boldsymbol{b};
\end{aligned}$$

Instead, if we refer to the R language, the previous operators can be defined more easily to the flexibility of the intrinsic command `rep()`.

As a matter of fact, the R correspondence is the following:

$$\begin{aligned}
\mathcal{E}[\boldsymbol{v}]^r &\Longrightarrow \texttt{rep(}\boldsymbol{v}, each = r\texttt{)};\\
\mathcal{R}[\boldsymbol{v}]^s &\Longrightarrow \texttt{rep(}\boldsymbol{v}, s\texttt{)};\\
\boldsymbol{a}\,\mathcal{J}\,\boldsymbol{b} &\Longrightarrow \texttt{cbind(}\boldsymbol{a}, \boldsymbol{b}\texttt{)}.
\end{aligned}$$

Thus, in $GAUSS$, the following procedure produces the matrix $\mathsf{A}(\mathcal{C})$ containing all the random samples.

---

[3] $GAUSS$ language is a copyright software distributed by Aptech System, Inc. (2004); we are referring to the version 6.0.25. R is an open-source software freely available from the R Development Core Team (2004); we are referring to the version 2.0.1. A programming environment which shares many similarities with R is S-plus, distributed by the Insightful Corporation, Seattle, Washington.

[4] Note that the elementwise operator **.*** between the p rows vector $\boldsymbol{v}$ and the matrix of $1's$ $ones(k, m)$ produces a (k, m) matrix whose k rows are replicates of the rows vector $\boldsymbol{v}$.

```
PROC GENERAG(m,n);
LOCAL vett,dim,samplemat,j;
vett=seqa(1,1,m); dim=m^n;
samplemat=reshape(vett,dim,1);
for j(2,n,1);
samplemat=
reshape(vec(vett'.*ones(m^(j-1),m)),dim,1)~samplemat;
endfor;
RETP(samplemat);
ENDP;
```

In the R language, the same result is accomplished by the following function.

```
GENERAR<-function(m,n){
serie=1:m;samplemat=rep(serie,m^(n-1));

for(j in 2:n)
{ samplemat= cbind(rep(rep(serie,each=m^(j-1)),
m^(n-j)), samplemat)}
return(samplemat)}
```

Although the R function seems considerably simpler than the $GAUSS$ procedure, we found that the computation times of $GAUSS$ are significantly shorter than R on the same platform; however, the need of manipulating matrices of increasing size might reduce this advantage[5].

Of course, the previous procedures can be immediately modified for generating the matrix of the corresponding probabilities; in this case, it is convenient to define an external *proc*, or a *function*, where the discrete probability distribution of X is well defined, for any fixed parameter θ. Then, we should add a line in the code just after the instruction `samplemat` to evaluate the probabilities.

[5] For instance, with a default setting, we get an "insufficient memory" message from $GAUSS$ when $m = 3;\ n \geq 14$ while R is yet operating.

6. Some experiences

In this section we apply the proposed algorithm to the computation of the exact probability distribution of two statistics, both generated by sample experiments from the discrete Uniform distribution.

6.1. Sampling distribution for the range

For instance, suppose that $X \sim Ud(m)$ is a discrete Uniform random variable on the support $\mathbb{S}_{\mathbb{X}} = \{x : x = 1, 2, \ldots, m\}$ so that:

$$Pr\,(X = x) = \frac{1}{m}, \quad x = 1, 2, \ldots, m.$$

Then, for $m = 6$ and $n = 3$, we require the exact distribution of the statistic *sample range* defined by:

$$T_n = max(X_1, X_2, X_3) - min(X_1, X_2, X_3)$$

where (X_1, X_2, X_3) is a random sample generated by X. In fact, the problem is equivalent to search for the exact distribution of the range of the points we get in the throwing of 3 fair dice.

Then, if we draw a random sample from X of size $n = 3$, all the possible samples in $\mathcal{C}$ consist of $6^3 = 216$ elements generated from the allocation with replacement of 6 distinct numbers in 3 urns.

Thus, after generating the 216 triples of t_n, we need to list only the distinct ones and compute the corresponding probabilities[6]. Finally, when $\mathbb{S}_{\mathbb{T}} = \{t : t = 0, 1, 2, \ldots, 5\}$, Figure 1 shows the computed sampling distribution of T_n for $n = 3, 4, \ldots, 8$.

Notice how the shape of the sampling distribution varies also at small changes in the sample size.

[6] Matrix languages have convenient commands for this; e.g., in GAUSS, the command `UNIQUE(`$\boldsymbol{v}$`,1)` selects just the distinct elements of $\boldsymbol{v}$.

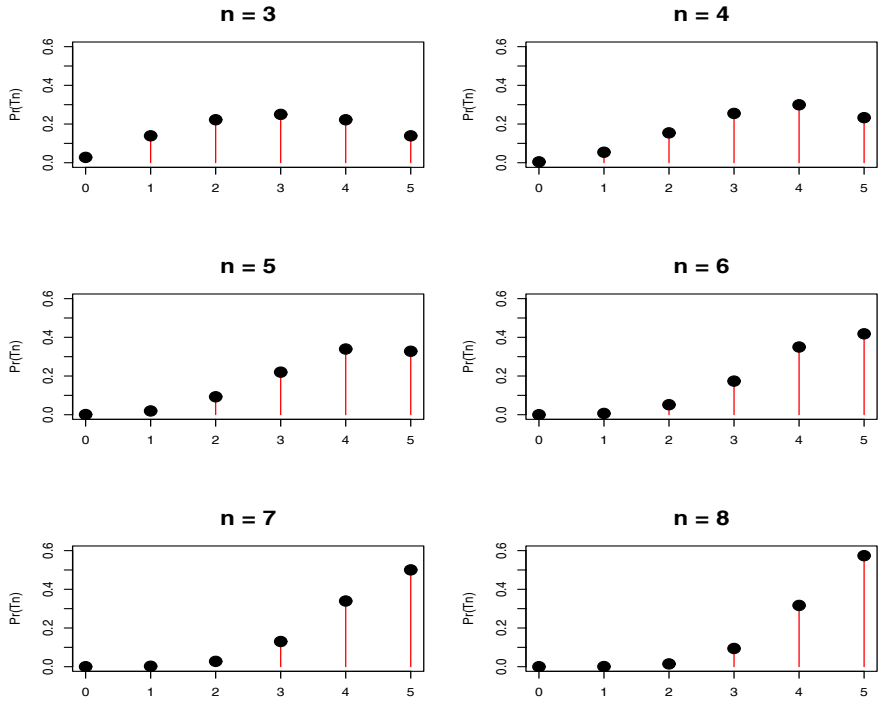

Figure 1. Exact sampling distributions of the range T_n when $X \sim Ud(6)$.

6.2. Sampling distribution of an index of diversity

In the socio-political literature, it has been discussed an index of diversity, strictly related to the well know Gini index[7]. The index is aimed to measure the number of relevant parties (or groups) in policy analysis of institutions (e.g. Parliament, Senate, etc.). Most of the current work in this area is limited to descriptive analysis and, generally, to the study of frequency distributions. Of course, this kind of measure might be

[7] Independently, we introduced this measure in Piccolo (1998, 153), and applied its normalized version in D'Elia and Piccolo (2005b) for interpreting the uncertainty in choosing from ordinal random variables. Since its range is greater than the Gini index and the entropy measures, this index could support an increased discrimination power.

introduced also as a diversity measure of discrete random variables.

For a discrete distribution, defined over the support $\mathbb{S}_{\mathbb{X}} = \{x : x = 1, 2, \ldots, m\}$, the index of Laakso and Taagepera (1979) is defined by[8]:

$$\mathcal{A} = \frac{1}{\sum_{x=1}^{m} \{Pr(X = x)\}^2}.$$

If the sampling experiment generates a random sample of size n, then the corresponding statistic is

$$T_n = \frac{n^2}{\sum_{x=1}^{m} N_x^2},$$

being N_x the absolute frequencies of the event $(X = x)$, $x = 1, 2, \ldots, m$.

For $X \sim Ud(m)$, we derive the exact sampling distribution of T_n when $m = 3$ and $n = 12$. In this case, the elements of the sample space $\mathcal{C}$ are $3^{12} = 531441$ and to each of them corresponds a probability of $1/3^{12} \simeq 1.882 \times 10^{-6}$. Moreover, the statistic is a function of (N_1, N_2, N_3) which is a Multinomial random variable whose probability distribution is:

$$Pr(N_1 = n_1, N_2 = n_2, N_3 = n_3) = \frac{12!}{n_1!\, n_2!\, n_3!} \frac{1}{3^{12}},$$

where $0 \leq n_i \leq 12$, $i = 1, 2, 3$; $\sum n_i = 12$.

---

[8] It is immediate to show that $\mathcal{A}$ varies between m when the random variable is a discrete Uniform, and 1 when it degenerates to a single value. Thus, a normalized expression on $[0, 1]$ is defined by considering $(\mathcal{A} - 1)/(m - 1)$.

It is also worth to note that, by analogy to the Box and Cox (1964) transformation, a generalized version of $\mathcal{A}$ defined by:

$$\mathcal{A}(\lambda) = \frac{\mathcal{A}^\lambda - 1}{\lambda},$$

produces the Gini (for $\lambda = -1$), the Simpson (for $\lambda \to 0$) and the Laakso and Taagepera (for $\lambda = -1$) indexes, respectively. We are currently studying what λ is optimal for inferential purposes.

Following the algorithm discussed in section 4, we found that the support of T_n is extremely variable depending upon the arithmetical combination of m and n. Then, if we sum over $\mathbb{S}_{\mathbb{X}}$ the constant probability $1/3^{12}$ we obtain the exact distribution of T_n.

To give an idea of the atypical shape of this kind of distribution, we show in the Figure 2 the exact probability distributions of T_n for $m = 3$ and varying $n = 3, 6, 9, 12$.

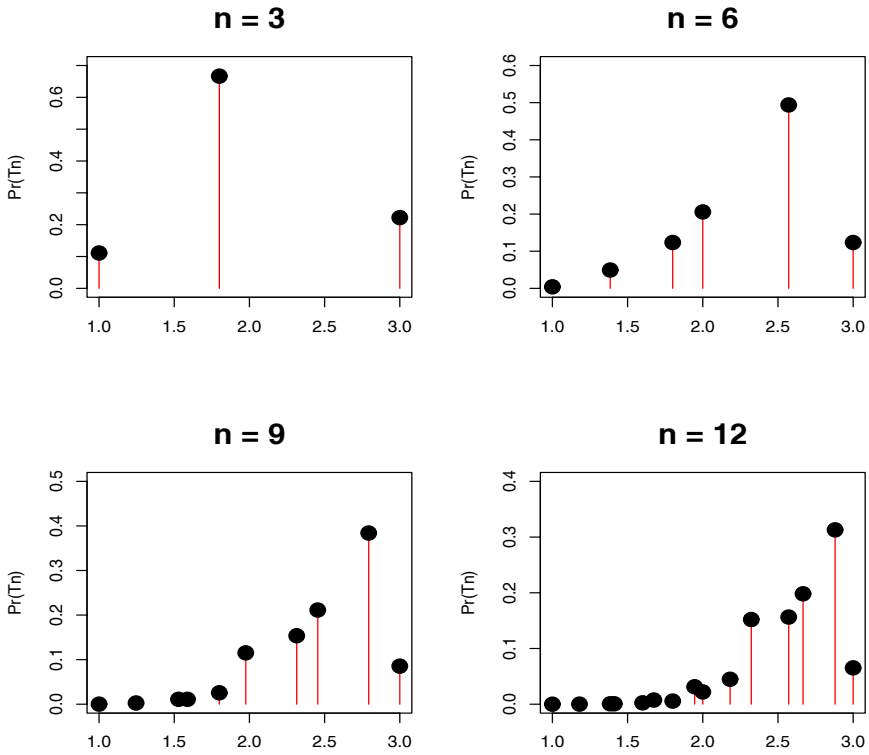

Figure 2. Exact sampling distributions of the index of Laakso and Taagepera when $X \sim Ud(3)$, for selected $n = 3, 6, 9, 12$.

7. Generalizations and concluding remarks

In general, the previous procedure should be modified for the support $\mathbb{S}_{\mathbb{X}}{}' = \{x : x = x_1, x_2, \ldots, x_m\}$. In fact, we need to establish the correspondence $i \longleftrightarrow x_i$, for any $i = 1, 2, \ldots, m$. In other words, the generation of the elements listed by the procedure are simply the indexes of the sample elements and *not* their values.

In this way, the matrix we obtain from the procedure is just the matrix of the indexes of the elements in the sample. The programming languages, mentioned in the previous sections, have effective commands to relate the indexes of a matrix to the corresponding element values. For instance, in the $GAUSS$ language, the command to select the elements in an arbitrary list (also with indexes repeated) is: `x[list]`, where `list` is an arbitrary column vector of real numbers (of course, rounded to the next integer).

Recently, D'Elia and Piccolo (2005a) applied successfully the approach of this paper to obtain the exact small sample distribution of the moment estimator of the parameter of the Inverse Hypergeometric random variable. In their work, the procedure allowed to compare effectively the exact distribution of the estimator with the Edgeworth and saddlepoint approximations, respectively.

Acknowledgements: This work benefit from the research structures of the Dipartimento di Scienze Statistiche, Università di Napoli Federico II, and of CFEPSR, Portici. We thank the referees for critical suggestions that significantly improved a preliminary version of the paper.

References

Aptech System, Inc. (2004), *GAUSS Mathematical & Statistical System*, version 6.0.25, User Guide and Language Reference I-II, Maple Valley, WA.

Box G. E. P. and Cox D. (1964), An analysis of transformations (with discussion), *Journal of the Royal Statistical Society*, Series B, 26, 211-246.

D'Elia A. and Piccolo D. (2005a), The moment estimator for the IHG distribution, in: (C. Provasi, editor) *Atti del IV Convegno S.Co.2005*, Bressanone, CLEUP Editrice, Padova, 245-250.

D'Elia A. and Piccolo D. (2005b), Uno studio sulla percezione delle emergenze metropolitane: un approccio modellistico, *Quaderni di Statistica*, 7, *this issue*.

Laakso M. and Taagepera R. (1979), Effective number of Parties: a measure with application to West Europe, *Comparative Political Studies*, 12, 3-27.

Piccolo D. (1998), *Statistica*, Edizioni il Mulino, Bologna.

The R Development Core Team (2004), *The R Project for Statistical Computing*, version 2.0.1, `http://www.r.project.org`